User's Guide

©2014, QNX Software Systems Limited, a subsidiary of BlackBerry Limited. All rights reserved.

QNX Software Systems Limited
1001 Farrar Road
Ottawa, Ontario
K2K 0B3
Canada

Voice: +1 613 591-0931
Fax: +1 613 591-3579
Email: info@qnx.com
Web: http://www.qnx.com/

QNX, QNX CAR, Momentics, Neutrino, and Aviage are trademarks of BlackBerry Limited, which are registered and/or used in certain jurisdictions, and used under license by QNX Software Systems Limited. All other trademarks belong to their respective owners.

Electronic edition published: Wednesday, July 30, 2014

Table of Contents

Preface: About this guide v
Typographical conventions vi
Technical support viii

Chapter 1: Overview of the Runtime for APK 9
Architecture 10
Applications and services in the QNX environment 11
Implementation 12
Adaptive partitioning for the Runtime for APK 14
Supported boards and APIs 15
Native libraries 16

Chapter 2: Developing apps for the APK runtime 19
Preparing APK files for the QNX platform 20
Converting APK apps to the BAR file format 21

Chapter 3: Using the Runtime for APK 23
Differences in app behavior 24
Controlling the APK navigation bar 25
Setting the screen dimensions 27
Setting the screen size and density 29
Enabling the debugger (adb) 30

Chapter 4: Compatibility and compliance 31

Preface
About this guide

This document describes the QNX Runtime for APK.

The QNX Runtime for APK is an add-on component for QNX platforms. It is not included in the main QNX installers (QNX CAR, QNX Apps and Media, and QNX SDP), but can be downloaded from the QNX web site.

To find out about:	See:
The runtime and how it is implemented on the QNX platforms	*Overview of the Runtime for APK* (p. 9)
Configuring the system to start the runtime	*Using the Runtime for APK* (p. 23)
How to package your apps for the QNX platforms	*Developing apps for the APK runtime* (p. 19)
How to use the runtime	*Using the Runtime for APK* (p. 23)
QNX Runtime for APK compliance and CTS results	*Compatibility and compliance* (p. 31)

Typographical conventions

Throughout this manual, we use certain typographical conventions to distinguish technical terms. In general, the conventions we use conform to those found in IEEE POSIX publications.

The following table summarizes our conventions:

Reference	Example
Code examples	`if( stream == NULL)`
Command options	`-lR`
Commands	`make`
Constants	`NULL`
Data types	`unsigned short`
Environment variables	***PATH***
File and pathnames	`/dev/null`
Function names	*exit()*
Keyboard chords	**Ctrl–Alt–Delete**
Keyboard input	`Username`
Keyboard keys	**Enter**
Program output	`login:`
Variable names	*stdin*
Parameters	*parm1*
User-interface components	**Navigator**
Window title	**Options**

We use an arrow in directions for accessing menu items, like this:

You'll find the Other... menu item under **Perspective** **Show View**.

We use notes, cautions, and warnings to highlight important messages:

Notes point out something important or useful.

Cautions tell you about commands or procedures that may have unwanted or undesirable side effects.

© 2014, QNX Software Systems Limited

Warnings tell you about commands or procedures that could be dangerous to your files, your hardware, or even yourself.

Note to Windows users

In our documentation, we typically use a forward slash (/) as a delimiter in pathnames, including those pointing to Windows files. We also generally follow POSIX/UNIX filesystem conventions.

© 2014, QNX Software Systems Limited

Technical support

Technical assistance is available for all supported products.

To obtain technical support for any QNX product, visit the Support area on our website (*www.qnx.com*). You'll find a wide range of support options, including community forums.

© 2014, QNX Software Systems Limited

Chapter 1
Overview of the Runtime for APK

The QNX Runtime for APK allows you to run APK applications on QNX platforms.

About the QNX Runtime for APK

The Runtime for APK is a QNX component used for executing applications that are packaged in the industry-standard `.apk` format. APK applications are commonly developed in Java and compiled for Dalvik. They take advantage of the rich Java API and high-performance graphics libraries such as OpenGL ES.

The large community developing APK applications means that there are many options for tooling and libraries, including generic libraries and vendor-specific APIs. The QNX Runtime for APK supports most APIs for the APK format. Only APIs that are irrelevant to QNX platforms, such as those specific to telphony or SMS, are not supported. For more detailed information, see "*Compatibility and compliance* (p. 31)".

The QNX Runtime for APK environment can be integrated into systems and HMIs running other technologies such as HTML5, Qt and OpenGL ES. See "*Architecture* (p. 10)" for details.

Supported apps

The QNX Runtime for APK supports the vast majority of applications written for Android devices; these APK applications can be run without modification on QNX platforms. However, apps that use libraries that are not in Google's Android SDK: Google Play Services, Google Maps, Google Voice, are not supported on QNX platforms. Use of these libraries is governed by an agreement with Google.

Demo apps

The following demo apps are included on QNX platforms:

- API Demos
- Calculator

The demo applications are present in the HMI if the Runtime for APK is running.

© 2014, QNX Software Systems Limited

Architecture

The QNX Runtime for APK component is an operating system that runs in conjunction with QNX Neutrino RTOS. It runs natively (not emulated) and is integrated similar to how other UI technologies are integrated within a QNX environment.

APK applications run on a register-based virtual machine, as opposed to the stack-based architecture of Java virtual machines. APK applications run on the Dalvik virtual machine which has been ported from Linux onto the QNX Neutrino RTOS.

When an APK application runs, the virtual machine converts the application code (.apk file format) to machine code (`.dex` file format) and optimizes it to reuse common code between applications. The final optimized code has an `.odex` file format. APK files may include native code as libraries (`.so`). These files will run in the QNX Runtime for APK *without* any recompiling.

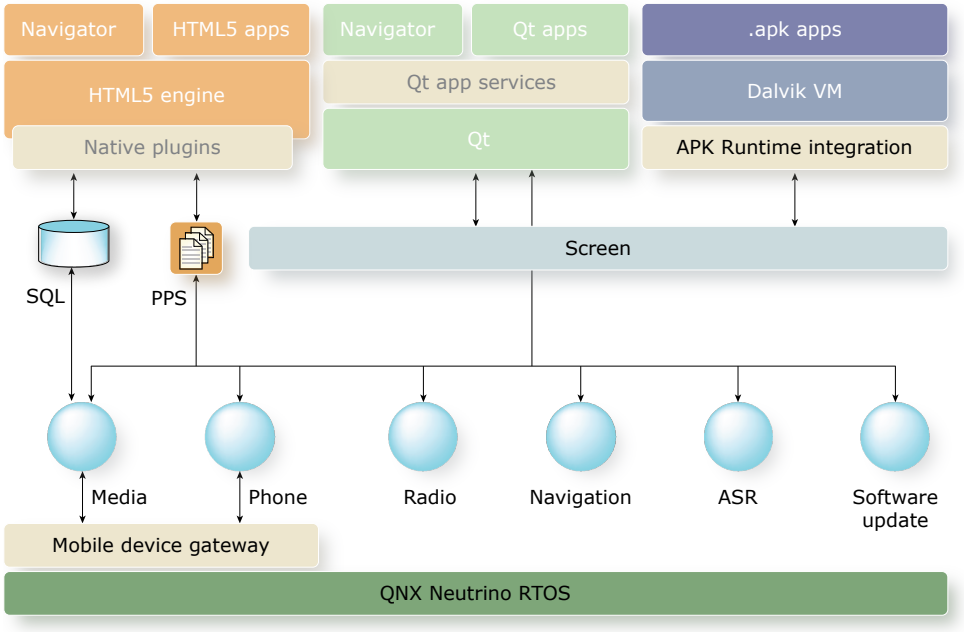

Figure 1: Typical implementation with the QNX Runtime for APK

The figure above shows how the QNX Runtime for APK fits into a typical QNX implementation, such as QNX CAR.

For more information about how the Runtime for APK is isolated from the rest of the QNX system, see "*Implementation* (p. 12)".

© 2014, QNX Software Systems Limited

Applications and services in the QNX environment

The QNX Runtime for APK supports APK applications relevant to the QNX environments.

The figure below shows in more detail how the Runtime for APK and APK applications fit into a QNX environment. Other technologies (HTML5 and Qt) are not shown. The grey boxes represent applications and services to which APK applications don't have access.

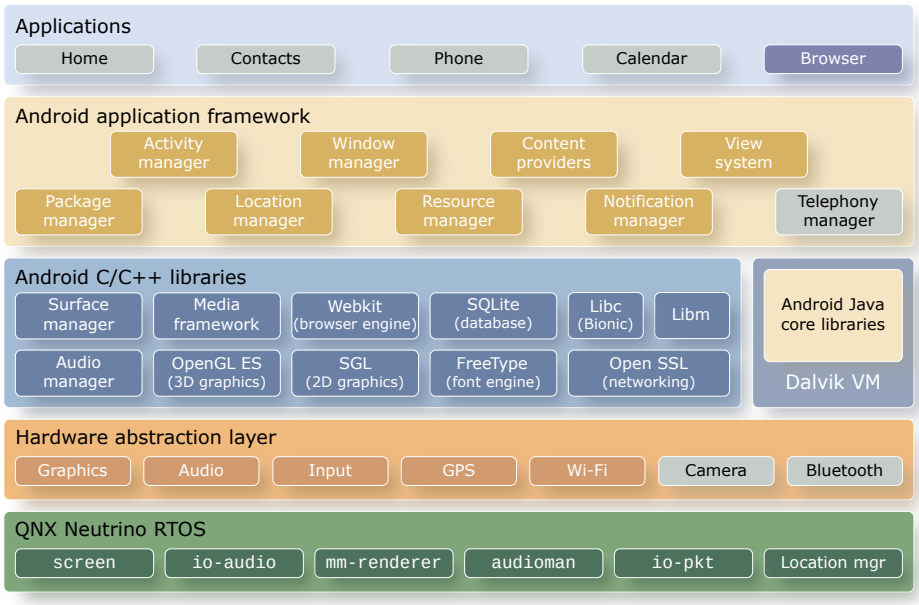

Figure 2: Detailed view of the Runtime for APK in the QNX environment

For more information about the applications and services currently not supported by the QNX Runtime for APK, and why they are not supported see "*Compatibility and compliance* (p. 31)".

© 2014, QNX Software Systems Limited

Implementation

You can use the Runtime for APK to run your `.apk` applications on QNX platforms.

Modifications

On a typical mobile device, the Dalvik virtual machine runs on Linux as a root user. The applications are sandboxed and the interfaces to device-specific resources, such as the camera, the phone and the Wi-Fi, are abstracted by the API.

On QNX platforms, the Runtime for APK isn't allowed to operate as root because this restriction ensures that the APK Runtime can't interfere with the QNX platform software. The Runtime for APK itself is sandboxed to protect the rest of the system from any unpredictable APK applications.

To allow Runtime for APK to operate normally while still protecting the QNX system, the following modifications have been made to how the Runtime for APK is implemented:

- The system maps the Runtime for APK into a special UID/GID range (800-999) and allows UID 800 (`AID_ROOT`) to act as root in specific parts of the filesystem.
- The runtime has a set of low-level libraries (for example, `libc`) named `libbionic`. QNX has reimplemented `libbionic` to map Linux system calls into Neutrino RTOS kernel messages where necessary.
- The `libbionic` library maps between system UID/GIDs and Runtime for APK UID/GIDs, so that a call to *getuid()* in the main Runtime for APK process returns root (0), which is actually 800 for the OS.
- Some actions of the QNX platform (for example, interactions with the screen, notifications regarding app installation, and so on) take place via updates to PPS objects. In some cases, Runtime for APK system elements have been modified to monitor these specific PPS objects for events.
- Adaptive Partitioning restricts the amount of CPU the Runtime for APK can consume (40% by default). This restriction ensures that sufficient CPU is available for other applications because the APK Runtime works on the assumption that it owns the CPU.
- The QNX platform creates file resources that are fundamental assumptions in the Runtime for APK. Specifically, `/data` and `/system` are process manager symlinks into the Runtime for APK sandbox.
- The QNX platform's `apk_launcher` starts the Runtime for APK binary under the control of the Application Launcher and the Authorization Manager to enforce specific `procmgr` abilities and file Access control lists (ACLs).

© 2014, QNX Software Systems Limited

Applications that request specific permission from the user during installation are automatically granted access. QNX platforms assume that applications have been approved for use in the customer environment, so there's no mechanism for denying permission requests.

Installed location

The Runtime for APK is installed as a native application, located in the`/apps/sys.apkruntime.gYABgKAOw1czN6neiAT72SGO.ns` directory. At runtime, symlinks in the `/data` and `/system` directories point into the sandbox for the Runtime for APK application (`/accounts/1000/appdata/sys.apkruntime.gYABgKAOw1czN6neiAT72SGO.ns/`).

Terminated applications

APK applications that have been dismissed by a user are terminated immediately and any system resources allocated are released.

© 2014, QNX Software Systems Limited

Adaptive partitioning for the Runtime for APK

The APK runtime uses about 110MB RAM plus whatever the applications consume.

The memory consumption of apps varies significantly, but each app is typically in the range of 5 to 50MB.

To make the most efficient use of system memory, the `apkruntime-start.sh` script creates adaptive partitions for the APK runtime. There are four partitions, with execution budgets (specified in `apkruntime-start.sh`) as follows:

Partition	Budget (%)
System (outside APK runtime)	60
Foreground (APK runtime)	40
Background	0
Stop	0

The total execution budget is always made available to APK runtime; therefore if system resources are available, APK runtime can consume up to 100% of the budget. If there are other processes or threads consuming system resources, you can be assured that APK runtime will still have at least 40% of the execution budget.

For more information about adaptive partitioning, see the *Adaptive Partitioning User's Guide*.

© 2014, QNX Software Systems Limited

Supported boards and APIs

The QNX Runtime for APK supports standard APK APIs, and can be run on several popular boards,.

API

Currently the QNX Runtime for APK uses the Jellybean (4.2.2) release, API level 17.

Boards

Freescale i.MX6Q SABRE Lite

Due to the limited memory available on this board, we recommend that during your evaluation of the Runtime for APK you carefully monitor your use of other high-resource applications. Support of this board requires the QNX 6.6.0 Graphics GA Patch. This patch is available from our *QNX Download Center*.

Texas Instruments OMAP5 uEVM (ES2 Silicon)

We strongly recommend that you use fastboot with EMMC or eMMC+SATA in conjunction with the Runtime for APK. Use of an SD card can result in a slow start-up.

- For more information about boot optimization, see the *Boot Optimization Guide*.
- Runtime for APK cannot be used on the Texas Instruments Jacinto 5 (J5 Eco Rev 5) board.

© 2014, QNX Software Systems Limited

Native libraries

The QNX CAR Runtime for APK makes available native libraries to apps that include native code.

About using native code

Apps written for the QNX Runtime for APK may include native Android code. After an `.apk` file has been built with the Android tools, it must be converted to a QNX compatible `.bar` file. (See "*Developing apps for the APK runtime* (p. 19)".) QNX has ported all the Android Linux libraries to the QNX platform, so any plugin code that requires these libraries will run.

Note that Google discourages the use of JNI plug-ins to access native libraries. Implementing parts of an app in native-code languages (using the NDK toolset) may be helpful if you are re-using existing code, but as is explained on the Android developer site at *https://developer.android.com/tools/sdk/ndk/index.html*, native code on Android generally doesn't result in a noticeable performance improvement, but it does increase the complexity of the app.

Libraries

The following native libraries are available:

Library	Notes
`libandroid.so`	Native Android activity support
`libc`	C library
`libdl`	Dynamic linker
`libEGL.so`	Native OpenGL surface management
`libGLESv1_CM.so`	OpenGL ES 1.0
`libGLESv2.so`	OpenGL ES 2.0
`libjnigraphics.so`	JNI graphics
`liblog`	Android logging
`libm`	Math library
`libOpenSLES.so`	OpenSL ES 1.0.1 audio support
`libOpenMAXAL.so`	OpenMAX AL 1.0.1 support
`libz`	Zlib compression

© 2014, QNX Software Systems Limited

Minimal support is available for C++.

© 2014, QNX Software Systems Limited

Chapter 2
Developing apps for the APK runtime

Applications can be developed just like other `.apk` apps, then re-packaged in the BAR (`.bar`) format for the QNX Runtime for APK .

To develop an APK app for QNX platforms, start by developing it exactly as you would for an Android-powered device:

- Follow the workflow outlined at *http://developer.android.com/tools/workflow/index.html*.
- Download the Android SDK from the developer site at *http://developer.android.com/sdk/index.html*.
- Follow the instructions on the Android *developer site* to set up the development environment, build the app, and run it on the emulator or on an Android-powered device to debug and test it.
- When you have finished developing and testing your app, you will need to validate and package it for QNX platforms.

The Android NDK is not required unless the app uses C/C++. For more information, see "*Native libraries* (p. 16)".

© 2014, QNX Software Systems Limited

Preparing APK files for the QNX platform

Before you can use the runtime for your APK applications, you must first repackage the applications into the BAR file format.

Packaging workflow

Once you have the debugged and tested your `.apk` app, and have it working as it should, you'll need to do the following:

1. Check the compatibility of the app for QNX platforms. To do this, use the `apk2bar` validation tool which comes with the Runtime for APK.

   This validation step is essential, because some APIs that are specific to mobile devices or SMS aren't relevant to QNX platforms, and hence are not supported.

2. When you have ensured that you app is compatible with QNX platforms, you can use the command-line tools provide with the QNX Runtime for APK to re-package your app in the BAR file format.

   There's no need to sign your BAR files to use with QNX platforms.

3. Copy the `.bar` app file to the QNX target board for installation. Once installed, you can test your app with the use of the debugger, `adb`.

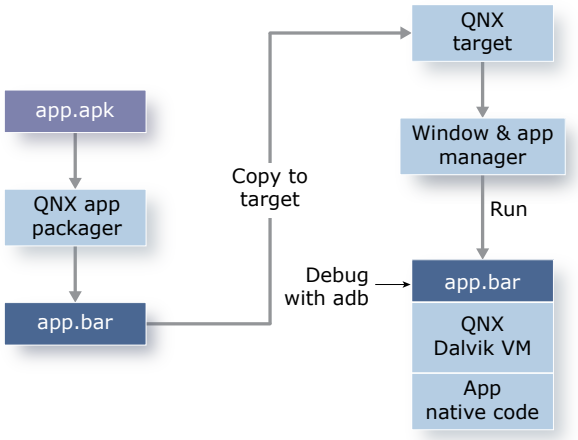

Figure 3: Workflow to prepare APK application

The diagram above shows the tasks involved in preparing APK applications to run on a QNX platform.

© 2014, QNX Software Systems Limited

Converting APK apps to the BAR file format

QNX platforms include a command-line tool for converting APK apps to the BAR file format.

To repackage your app in the BAR file format so you can use it on the QNX platforms, you should use the `blackberry-apkpackager` command-line packaging tool. This tool is provided with the QNX platform.

Not all APIs are supported, so you may need to modify your app before you can run it on QNX platforms. The `blackberry-apkpackager` packaging tool provides a list of exceptions which you may need to address. There may be entries in the BAR file that you need to fix manually.

See the chapter "*Compatibility and compliance* (p. 31)", and the documentation included with the tools for more information.

Here's a short summary or what you need to do to repackage your app in the APK format to the BAR file format:

1. Run `blackberry-apkpackager` to repackage your app.

   For example: `blackberry-apkpackager myapp.apk`

   This tool generates a list of exceptions, which you should review and address if necessary to ensure that your app is compatible with the platform.

   After addressing any exceptions, you have a compatible app in BAR file format.

   Note that you don't have to sign your BAR files for QNX platforms.
2. Transfer this BAR file to the target and install it using the `bar-install.sh` script.
3. Test your app on your target. Even if the tools didn't identify any significant issues, you may need to make further changes. The debugger, `adb`, is included with the runtime. It may help you identify any remaining problems. See " *Enabling the debugger (adb)* (p. 30) " for more information.

© 2014, QNX Software Systems Limited

Chapter 3
Using the Runtime for APK

The Runtime for APK and associated demo applications are preinstalled, but the runtime may be disabled by default, depending on the platform.

If the Runtime for APK is not enabled by default, you can enable it on supported platforms.

Enabling the Runtime for APK

To enable the Runtime for APK, edit the file `/var/etc/services-enabled` to include the following line:

```
APKRRUNTIME:true
```

and then restart the system.

When the APK runtime is running, the following processes are present:

```
1974323 system/bin/init  init
2154570 system/bin/logd  logd
2191440 system/bin/binder  binder
2322459 system/bin/servicemanager  /system/bin/servicemanager
2322498 system/bin/mediaserver  /system/bin/mediaserver
2322510 system/bin/installd  /system/bin/installd
2326596 system/bin/keystore  /system/bin/keystore /data/misc/keystore
4091985 system/bin/logcat  /system/bin/logcat -f /data/log/death.log -v time -r 256 -n 4
```

The debugger (`adb`) is not enabled by default (and is hence not in the above list). For more information, see "*Enabling the debugger* (p. 30)".

The `apkruntime-start.sh` script must specify the appropriate screen dimensions for the runtime to use to render its applications. For information about why this is so and for instructions on how to set the screen resolutions and dimensions, see "*Setting the screen dimensions* (p. 27)".

Uility scripts

The Runtime for APK on QNX platforms includes these utility scripts:

`ps-apkruntime-core.sh`

Handy for checking whether the runtime is running.

`ps-apkruntime-apps.sh`

See what's currently running.

These scripts are located at `apps/sys.apkrun time.gYABgKAOw1czN6neiAT72SGO.ns/native/scripts`.

© 2014, QNX Software Systems Limited

Differences in app behavior

APK applications on QNX platforms may look and behave differently than on mobile devices.

Differences in app behavior on QNX platforms

Here are the main differences using the QNX CAR platform as an example:

- The APK action bar is below the QNX CAR information bar, so icons or notifications in the action bar don't appear at the top of the screen.
- The APK navigation bar is above the QNX CAR navigation bar. See “Controlling the APK navigation bar”.

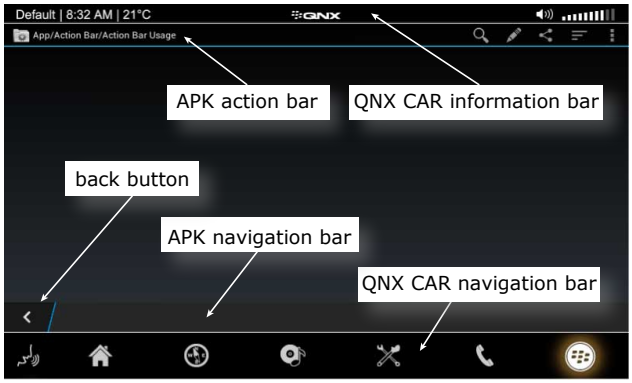

Figure 4: The APK action and navigation bars and the QNX information bar

© 2014, QNX Software Systems Limited

Controlling the APK navigation bar

By default, the APK runtime ensures that applications can't dismiss the navigation bar and its **back** button.

About the APK navigation bar

APK apps use the “back” action in the same way that a **back** button is used on a web browser. On many mobile devices (phones and tablets) the “back” function is implemented by a dedicated hardware (physical) button on the device. On the QNX platforms, this **back** button appears in the APK runtime navigation bar. This navigation bar is located above the QNX tab/navigation bar at the bottom of the screen. It has only a single icon: the **back** button is used to inititiate the **back** action.

A system built with a QNX platform might not include a hardware **back** button. By default, therefore, the APK runtime ensures that applications can't dismiss the APK runtime navigation bar and its **back** button, because with neither a hardware button nor a screen button, the user can no longer initiate the “back” action.

Showing and hiding the APK navigation bar

The default setting that locks the APK navigation bar on the screen is controlled by a parameter in the APK runtime configuration file:

```
//apps/sys.apkruntime.gYABgKAOw1czN6neiAT72SGO.ns/native/system/init.cfg
```

In this file, the line:

```
setprop car.lock.NavBar true
```

ensures that the APK navigation bar can't be dismissed.

If your implementation needs to allow applications to dismiss the APK runtime navigator bar, you can set this configuration parameter to `false`:

```
setprop car.lock.NavBar false
```

Once you've unlocked the APK navigation bar, the presence of the bar can be controlled by sending `SWIPE_DOWN` commands to the APK Runtime control PPS object.

For example, if the APK navigation bar is displayed, sending `echo "msg::SWIPE_DOWN"` to the `/pps/services/apkruntime/control` PPS object will hide the APK navigation bar. Sending a second `echo "msg::SWIPE_DOWN"` will display the APK navigation bar again.

Using the `SWIPE_DOWN` command will result in a bar at the *top* of the application display area. This bar contains a **Hide Bar** icon or a **Pin Bar** icon, depending on whether the APK navigation bar is displayed or hidden. Touching these icons hides or shows the APK navigation bar.

© 2014, QNX Software Systems Limited

With neither a hardware button nor a screen **back**, the user can no longer initiate the “back” action. This may leave an application running with no way for the user to properly navigate within the app.

© 2014, QNX Software Systems Limited

Setting the screen dimensions

The `apkruntime-start.sh` must specify the appropriate screen dimensions for the APK runtime to use to render its applications.

About the APK runtime screen

At system startup, System Launch and Monitor (SLM) launches the APK runtime by invoking a script named `/scripts/apkruntime-start.sh`. This script sets a number of properties for the APK runtime, including adaptive partitions (see "Adaptive partitioning for the APK runtime") and the screen resolution.

For example, on the QNX CAR platform, the usable screen size for an APK application is less than the physical screen size. This is because the default HMI divides the screen into three areas: the status (or information) bar, the application area, and the tab (or navigation) bar.

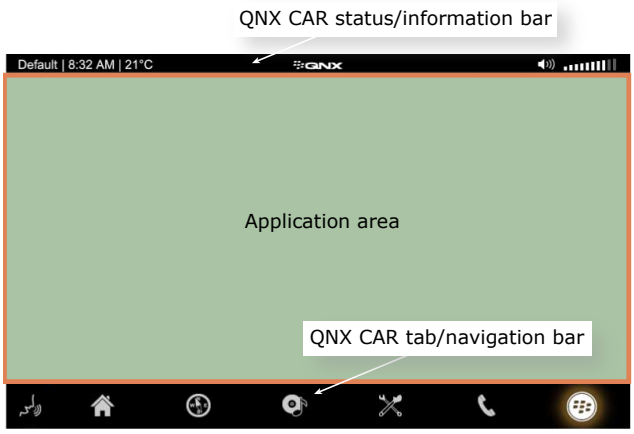

Figure 5: The QNX CAR platform screen areas

When an application is launched, its viewable area is contained within the application area. Different vendors may decide to use different heights for the status bar or tab bar, or even remove the status bar entirely.

Specifying screen dimensions in the startup script

Due to the variability of the application area on the screen, the `apkruntime-start.sh` startup script must specify the appropriate screen dimensions for the APK runtime to use to render its applications. The script specifies values for the `WIDTH` and `HEIGHT` variables that it sends to the the PPS `control` object (found at `/pps/services/launcher/control`). The `apkruntime-start.sh` script

© 2014, QNX Software Systems Limited

contains the following line, which sends the PPS message to set the variables and start the APK runtime:

```
echo "msg::start\ndat::$AP_APP,ORIENTATION=0,WIDTH=800,HEIGHT=395\nid:n:3080\nmsg::start\n" >> /pps/services/launcher/control
```

You should ensure that these are the correct dimensions for your screen and its components, and then adjust them as needed.

If `WIDTH` and `HEIGHT` are not provided, APK applications take up the entire screen. Even if you chose not to set these variables, APK applications will be overlaid by the status bar and tab bar because of their z-ordering.

The `set-resolution.sh` script automatically adjusts the `apkruntime-start.sh` script (for example, if you change the resolution with a `set-resolution.sh 720p` command).

© 2014, QNX Software Systems Limited

Setting the screen size and density

The Runtime for APK supports different screen sizes.

Depending on the display selected for your implementation, you may want to change the default APK Runtime screen appearance by changing the screen density.

To change the default screen density, in the APK configuration file:
`/apps/sys.apkruntime.gYABgKAOw1czN6neiAT72SGO.ns/native/system/init.cfg`

set the configuration parameter `setprop ro.sf.lcd_density` to the desired screen density, in pixels per inch. For example, the following configuration sets the screen density to medium density (approximately 160 pixels per inch):

```
setprop ro.sf.lcd_density 160
```

The table below lists supported screen densities and the corresponding configuration value.

Screen size	Value
LDPI (low density)	`120`
MDPI (medium density)	`160`
HDPI (high density)	`240`
XDPI (extra-high density	`320`

© 2014, QNX Software Systems Limited

Enabling the debugger (adb)

By default, the Android Debug Bridge (adb) is included on a QNX platform with the Runtime for APK.

adb enables loading, execution, breakpoints, and single-stepping through application code. If you want to use adb, but it is not already started, you need to edit the build configuration file and then restart your target.

To start adb:

1. Check whether the file *apk_app_path*/native/system/build.cfg contains the following line:
   ```
   persist.service.adb.enable=1
   ```
   Note that *apk_app_path* is the path to the APK runtime application, for example:
   ```
   /apps/sys.apkruntime.gYABgKAOw1czN6neiAT72SGO.ns/
   ```
2. Edit the build.cfg file to enable the debugger.
   - If the "enable" line isn't present, add it using the following command:
     ```
     "persist.service.adb.enable=1" >>
     <apk_app_path>/native/system/build.cfg
     ```
   - If the "enable" line is present, edit it to ensure that the value is set to 1.
3. Restart the target.

© 2014, QNX Software Systems Limited

Chapter 4
Compatibility and compliance

The QNX Runtime for APK doesn's support all APK functionality because some are not relevant to the QNX platform.

About Android compatibility

Google defines Android compatibility in its Compatibility Definition Document (CDD), and a compatibility test suite (CTS) that can be used to assess implementations of platforms supporting APK applications. This suite comprises some 18,000 automated tests, plus a number of manual tests.

At present the CDD and CTS are oriented to the needs of tablets and smartphones. Therefore, not all compatibility requirements are relevant to QNX platforms, which are designed for automotive or general embedded systems. Features typical of tablets and smartphones may not be supported because:

- the required hardware sensors, such as a barometer, gyroscope or light sensor aren't supported
- the features or behaviours tested are not relevant to the QNX platform

This disparity between the CDD and CTS specifications and the requirements the QNX platform is designed to meet means that the QNX Runtime for APK isn't able to achieve a 100% success rate with the CTS. The success rate is currently 95% overall, and 97% for tests relevant to the QNX platform (telephony-specific features, personal information management (PIM) removed).

CTS exclusions

The QNX Runtime for APK doesn't support all capabilities available for APK applications. Some capabilities may be unsupported for reasons such as unsupported services, settings, permissions, or hardware or software features.

QNX platforms don't support telephony-specific features such as SMS. QNX has also customized a number of CTS tests to take into account the specifics of the QNX platform. In order to understand correctly the results of these tests, you should review them with consideration to the lists of unsupported capabilities, modified tests, exclusions, and other anomolies. For detailed information about the CTS results, please contact your *QNX representative*.

Currently, QNX platforms don't support the following APK capabilities:

Accessibility

© 2014, QNX Software Systems Limited

These CTS tests are excluded because they are not relevant to the HMI of QNX platforms.

Bluetooth

The CTS tests presume that Bluetooth is used to connect a smartphone with a slave device (e.g., a telephone headset). However, QNX platforms are not smartphones and therefore, these tests are irrelevant.

Calendar

As a sandboxed environment on the QNX system, an Android calendar is not created.

Contact management

As a sandboxed environment on the QNX system, an Android contacts database is not created.

Launcher/Home screen

Launching of APK applications are controlled by the HMI on QNX platforms.

Renderscript

The QNX Runtime for APK doesn't support Renderscript at this time.

Telephony

QNX platforms are not smartphones and don't need to offer telephony services; therefore they may run on boards that lack the hardware required to support telephony.

Tests for applications or features that require these capabilities will fail.

Testing application compatibility

The compatibility tests generate five levels of warnings, from mild (Level 1) to severe (Level 5). If an application receives no warnings or only Level 1 warnings, then it is considered compatible with the QNX platform. You will still need to perform additional verification to make sure the application will run on your specific platform.

Applications that receive warnings of Level 2 or higher are generally considered incompatible, and will not run successfully on QNX platforms.

© 2014, QNX Software Systems Limited

Index

A

adaptive partitioning 14
APIs, supported 31
APK 31
    Customized CTS tests 31
APK application 32
    compatibility 32
APK runtime 9, 10, 14, 15, 21, 23, 24, 25, 27, 29, 30, 31
    See also Runtime for APK
    adaptive partitioning for 14
    adb 21, 30
    APIs, supported 31
    back 25
    demo apps 9, 23
    differences in app behavior 24
    enabling 23
    implementation 10
    memory consumption 14
    navigation bar 25
    overview 10
    processes present 23
    screen 27
    screen size and density 29
    supported boards 15
    tools for repackaging apps 21
        See also Runtime for APK
apk2bar 20
    verification tool 20
apkruntime-start.sh 14
app behavior, differences from standard mobile devices 24
apps 9, 12, 20, 21, 24, 31
    checking compatibility of 21, 31
    developinging for QNX Runtime for APK 20
    differences in behavior 24
    dismissed, running in the background 24
    permissions for 12
    repackaging 20, 21
        tool for 21
        tools for 20
    supported 9
    testing 31

B

BAR file 20, 21
    configuring 20
    converting apps to 20
    installing 21
blackberry-apkpackager 21
boards 15
    APK runtime support 15

C

compatibility 32
    APK application 32
    testing for 32
CTS 31
    customized tests 31
customized tests 31
    APK applications 31

D

Debug Bridge 30
debugger (adb) 21, 30
    enabling 30
demo apps 9, 23
    unresponsive if APK runtime isn't running 23

G

Google 9
    libraries 9

L

libbionic 12
libraries 9
    Google 9

M

memory 14
    consumption with APK runtime 14

N

navigation bar 25
    APK 25
    showing and hiding 25

P

processes present when APK runtime is running 23

Q

QNX CAR platform 12, 21, 24
    checking compatibility of apps 21
    differences in app behavior 24
    modifications to Runtime for APK 12
QNX platform 20
    repackaging APK apps for 20
QNX Runtime for APK 20
    developing APK apps for 20

© 2014, QNX Software Systems Limited

R

register-based virtual machine 10
Runtime for APK 12, 13
    access control lists 12
    adaptive partitioning for 12
    memory consumption 12
    modifications on QNX platform 12
    procmgr abilities 12
    running as root 12
    terminated applications 13

S

screen 27, 29
    APK runtime 27
    size and density in APK runtime 29
services-enabled file 23
supported 9
    apps 9

T

Technical support viii
terminated applications 13
    in Runtime for APK 13
tool for repackaging apps 21
Typographical conventions vi

U

unsupported capabilities 31
    APK 31

V

virtual machine 10
    register-based 10

W

warnings for supported APIs 31

© 2014, QNX Software Systems Limited